CAS FEE

CLEAN CODING IN JS

Markus Stolze

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL
FHO Fachhochschule Ostschweiz

Clean Coding - Gegenbeispiel

So nicht

```
async function helper(n, h) {
    let r = await pushResultToServer(n, h);
    let r1 = 0;
    if(r[1]) {
        r1 = 1;
    }
    else if(r[1] === false) {
        r1 = -1;
    }

    return r1;

}
```

- **Funtionsname zu generisch**
- **Parameter-Namen nicht sprechend**
- **Namen der lokalen Variablen nicht sprechend**
- **Schlecht durchdachte Verzweigung (“smell” myVar == true)**

Clean Coding

- **Thema bekannt gemacht durch Bücher von Robert C. Martin**
  - Clean Code *(Robert C. Martin, Pearson 2008)*
  - The Clean Coder: A Code of Conduct for Professional Programmers. *(Robert C. Martin, Pearson 2011)*
  - Clean Architecture: A Craftsman's Guide to Software Structure and Design. *(Robert C. Martin, Pearson 2017)*

Clean Code: A Handbook of Agile Software

The Clean Coder: A Code of Conduct for Professional

Clean Architecture: A Craftsman's Guide to Software

- **“Clean” = leicht verständlich (Code, Architektur, Dokumentation, Prozesse) != “Smelly”**
- **Verbunden:**
  - “Coding Conventions”, Refactoring, Tech Dept: (Auto) Code Formatierung, Coding Styles & Linters, …
  - SE Prinzipien: YAGNI, Single-Responsibility, Convention over Configuration, …
  - “Design Patterns”: MVC, Factory, Strategy, …
  - Coding Practices: TDD, BDD

Clean Coding

Clean Code: A Handbook of Agile Software

- **“Clean-ness” is in the eye of the beholder (or author, team, or community)**
- **Meaningfull Names**
- **Functions**
- **Comments**
- **Formatting**
- **Objects & Data Structures**
- **Error Handling**
- **Boundaries**
- **Unit Tests**
- **…**
- **Concurrency**
- **…**

- **Smells & Heuristics**
  - **Names**
    - N1: Choose Descriptive Names
    - N2: Choose Names at the Appropriate Level of Abstraction
    - N3: Use Standard Nomenclature Where Possible [-> Glossary, Domain Model]
    - N4: Unambiguous Names
    - N5: Use Long Names for Long Scopes [Function scope: shorter names are ok]
    - N6: Avoid Encodings [No type or scope info in name]
    - N7: Names Should Describe Side-Effects.
  - **Functions**
    - F1: Too Many Arguments
    - F2: Output Arguments
    - F3: Flag [boolean] Arguments [-> split]
    - F4: Dead Function [or any dead code]

Clean Coding

Clean Code: A Handbook of Agile Software

- "Clean-ness" is in the eye of the beholder (or author, team, or community)
- Meaningfull Names
- Functions
- **Comments**
- **Formatting**
- **Objects & Data Structures**
- **Error Handling**
- **Boundaries**
- **Unit Tests**
- …
- Concurrency
- …

- **Smells & Heuristics**
  - **Comments**
    - C1: Inappropriate Information
    - **C2: Obsolete Comment**
    - C3: Redundant Comment
    - C4: Poorly Written Comment
    - C5: Commented-Out Code
  - **Tests**
    - T1: Insufficient Tests
    - T2: Use a Coverage Tool!
    - T3: Don't Skip Trivial Tests
    - T4: An Ignored Test Is a Question about an Ambiguity [Document ambiguous requirements as ignored tests]
    - T5: **Test Boundary Conditions**
    - T6: Exhaustively Test Near Bugs
    - T7: Patterns of Failure Are Revealing
    - T8: Test Coverage Patterns Can Be Revealing
    - T9: Tests Should Be Fast

Clean Coding

Clean Code: A Handbook of Agile Software

- "Clean-ness" is in the eye of the beholder (or author, team, or community)
- Meaningfull Names
- Functions
- Comments
- **Formatting**
- **Objects & Data Structures [-> OO]**
- **Error Handling [no null returns or arguments]**
- **Boundaries**
- Unit Tests
- **…**
- Concurrency
- **…**

- **Smells & Heuristics**
  - **General [selection]**
    - G2: Obvious Behavior Is Unimplemented [function name is a contract!]
    - G5: Duplication [->DRY]
    - G8: Too Much Information [Exposing private parts]
    - G10: Vertical Separation [define vars/fun close to use!]
    - G14: Feature Envy [and other OO smells]
    - **G19: Use Explanatory Variables [yes, do this!]**
    - G21: Understand the Algorithm [avoid un-obvious code!]
    - G22: Make Logical Dependencies Physical [e.g. pure fns!]
    - G24: Follow Standard Conventions
    - **G25: Replace Magic Numbers with Named Constants**
    - G28: Encapsulate Conditionals [Use named conditions]
    - G34: Functions Should Descend Only One Level of Abstraction

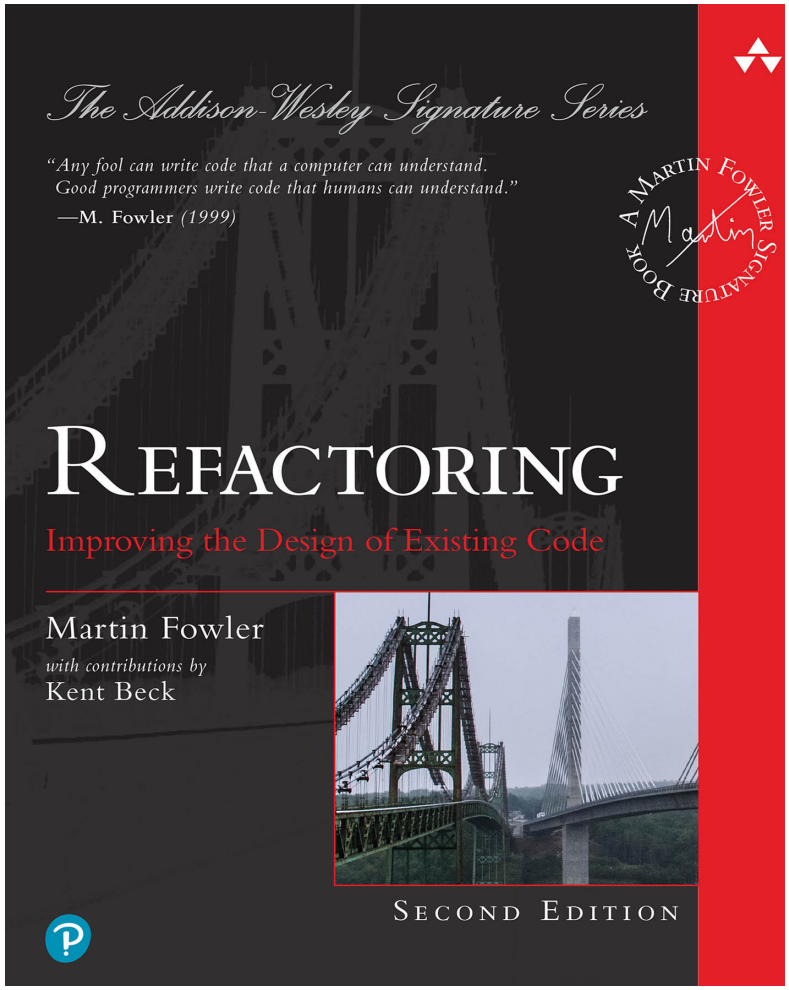
Refactoring

Chapter 3 Bad Smells in Code

- **Mysterious Name**
- **Duplicated Code**
- **Long Function**
- **Long Parameter List**
- **Global Data**
- **Mutable Data**
- Divergent Change
  [Not single responsibility]
- Shotgun Surgery
- **Data Clumps**
  [Turn them into objects]
- **Primitive Obsession**
  [Turn them into objects]
- Repeated Switches
  [E.g. if (online)]
- **Loops**
  [Instead: forEach, map, …]
- **Comments**
  [Try first to improve code]

[OO related Smells]

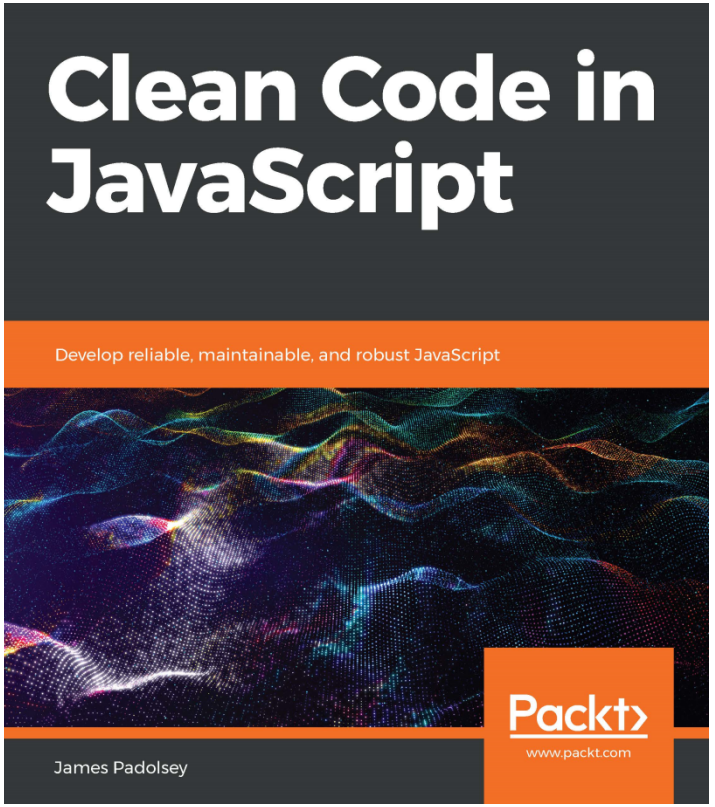
- Feature Envy
  [Missing cohesion]
- Lazy Element & Speculative Generality
  [You realize too late: YAGNI]
- Temporary Field
  [Conditional field -> split]
- Message Chains & Middle Man
  [Reduce length of object retrieval and delegation chains]
- Insider Trading
  [Too close coupling of classes]
- Large Class
- Alternative Classes with Different Interfaces
- Data Class
- Refused Bequest



Clean Coding in JS

Section 1: Introduction

- **Enemies of Clean Code**
  - **JavaScript [!]**
  - Management (Pressure to ship, bad metrics, **lack of ownership**)
  - **Self** (**showing off, stubborness**, …)
  - **Cargo Cult [!]**
- **Principles**
  - **SOLID (**Single Responsibility, **Open/Closed, …,)**
  - **…**
- **Balanced Abstraction**
- **Functional Programming Principles**
  - **Functional Purity [idempotence: same input -> same output]**
  - **Immutability**



Section 2: JavaScript

Clean Coding in JS

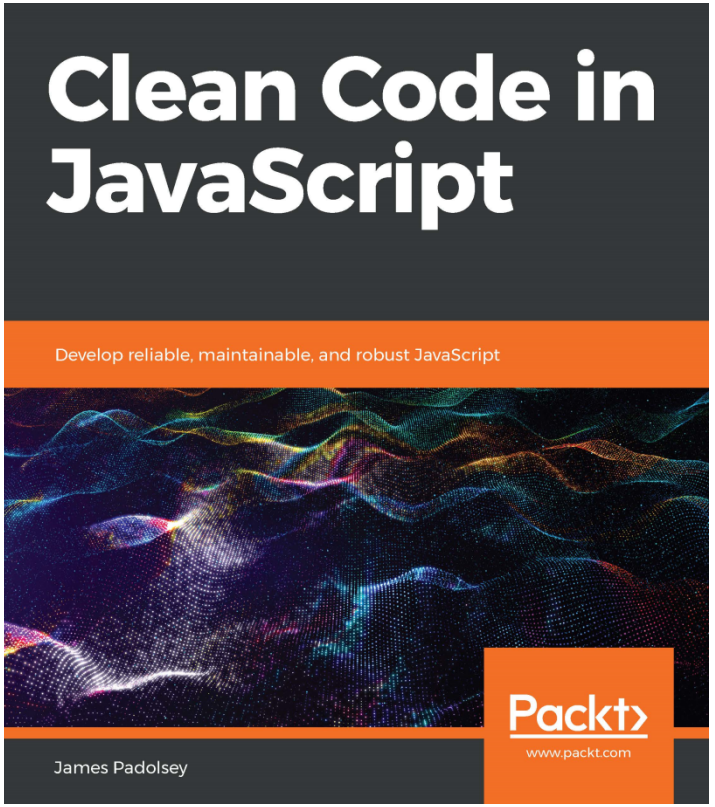
Section 2: JavaScript

Section 3: Crafting Abstractions

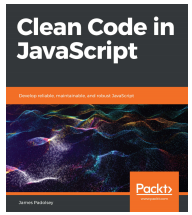
- **Desgin Patterns**
  - **MVC**
  - Modules
  - …
- **Real World Challenges**
  - **DOM binding & reconsiliation (event vs. state oriented) [-> next]**
  - **…**

Section 4: Testing & Tooling

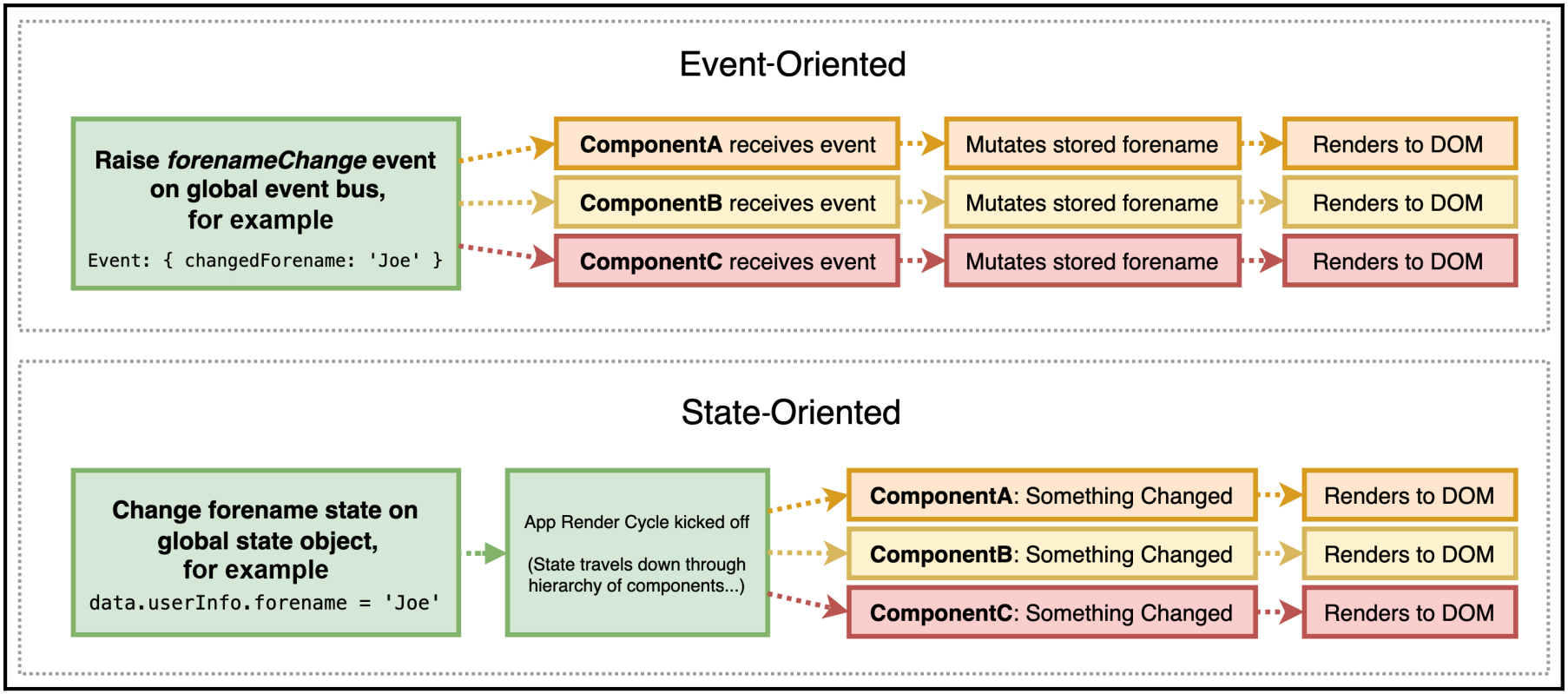
- **Types of tests: Unit, Integration, E2E**
- **Writing clean tests**
- **Tools for cleaner code: Linters & formatters, static typing, E2E Testing tools, Automated Build & CI,**
- **…**



Clean Coding in JS

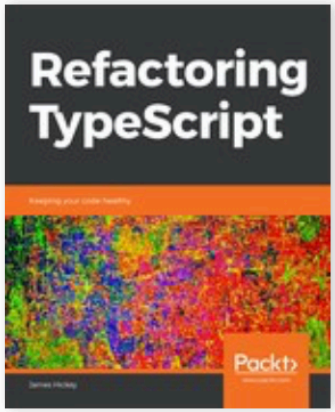


DOM binding & reconsiliation (event vs. state oriented)



Refactoring Typescript

- **Null Checks Everywhere**
- **Wordy Conditionals**
- **Nested Conditionals**
- **Primitive Overuse [use Classes!]**
- **Lengthy Method Signatures**
- **Methods that Never End [too long code]**



Refactoring TypeScript

0 REVIEWS

by James Hickey

Publisher: Packt Publishing

Release Date: October 2019

ISBN: 9781839218040

Topic: Refactoring

Clean Code: Empfehlungen

- **Code Wars -> Skills! ✓**
- **Bonus Regel:**
  State Machines statt mehrere Booleans
- **Prettier !**
- **Coding styleguide festlegen: AirBnb?**
- **Coding Styleguide Check automatisieren**
  ESLint Rules: AirBnb?
  - Lokal
  - Check-In
- **Framework best Practices “studieren”**
- **Gegenseitige Code Reviews**
  - Pull Requests für den check-in?

INPUT DER GRUPPEN

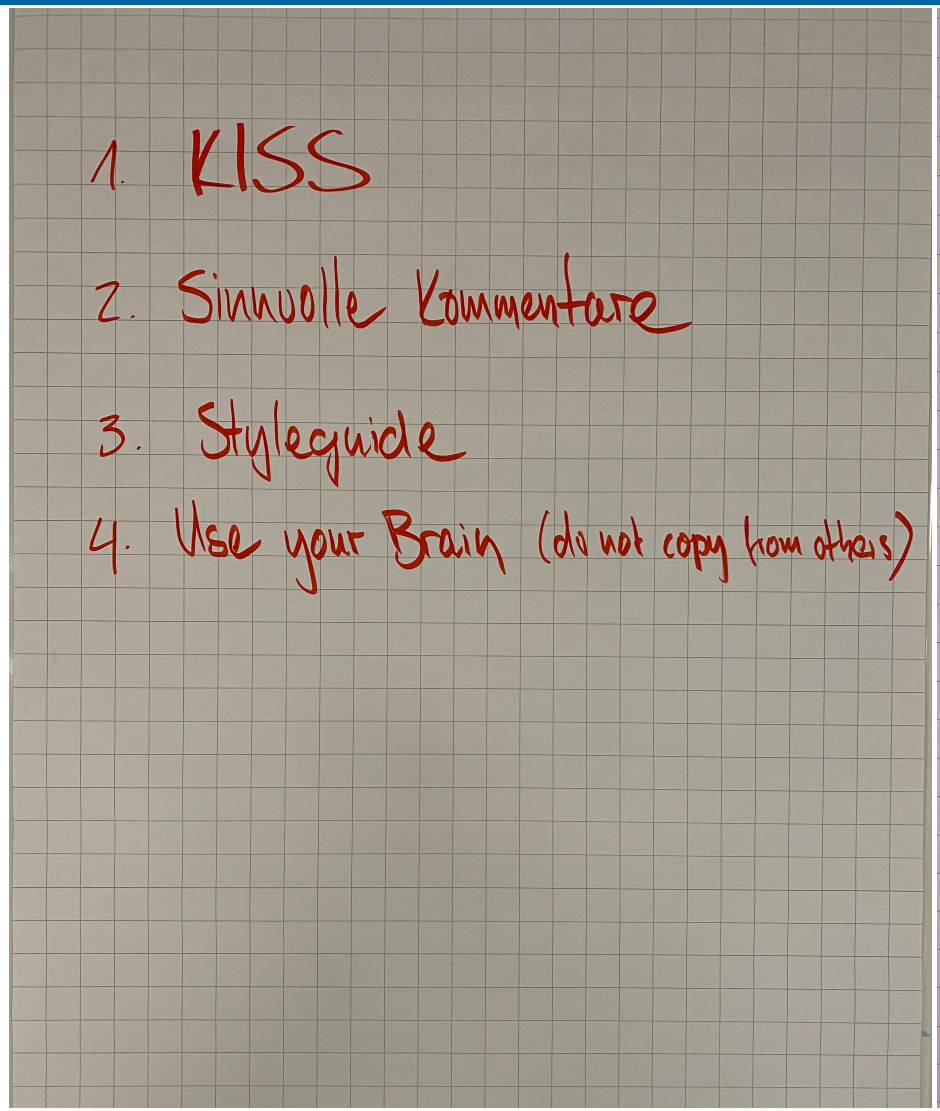
Flips 1

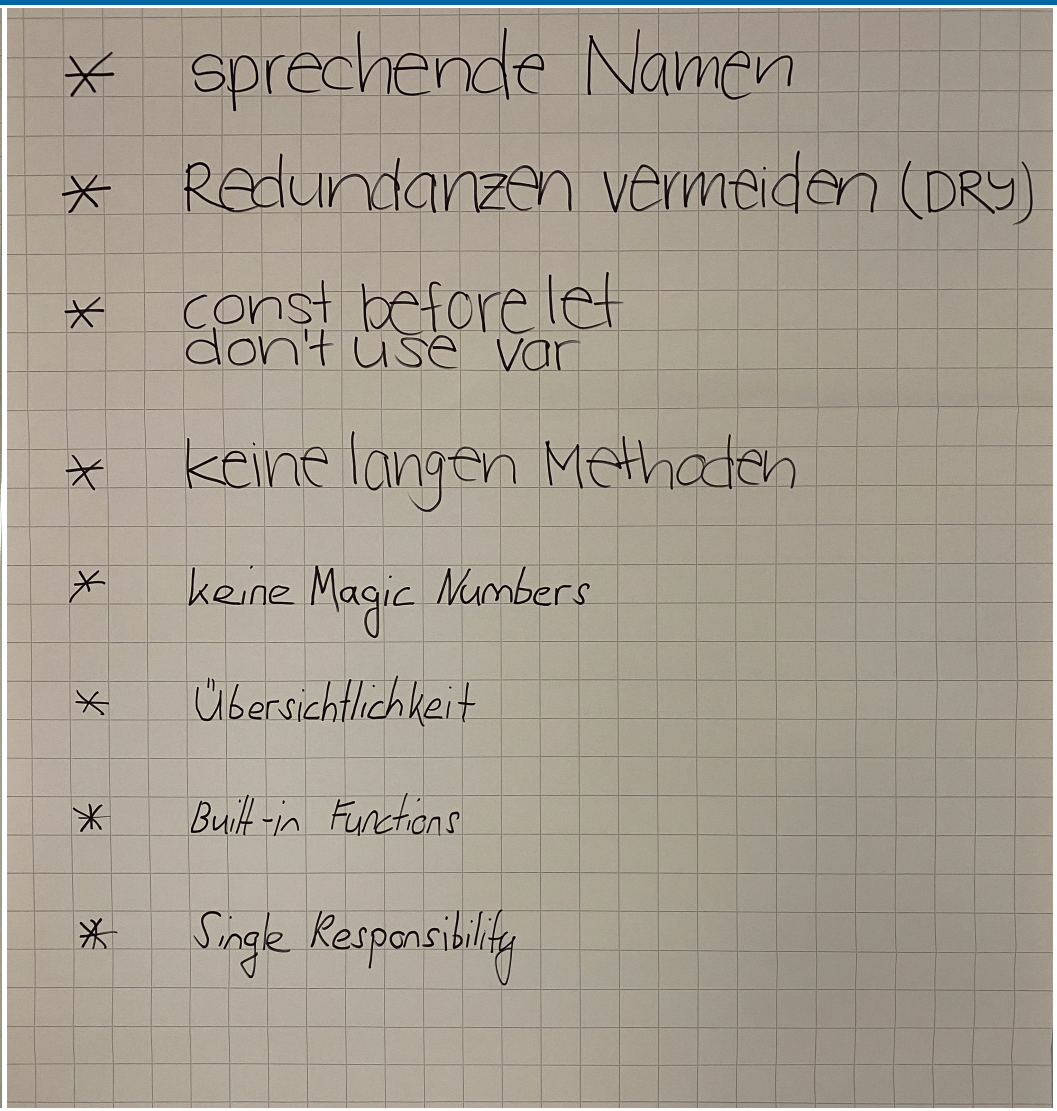
- Sprachfeatures nutzen
  - → let, const, 'array-function'
- Verständliche Conditionals
  - → if (!false) ... else
- Lesbarkeit
  - → so kurz wie möglich, aber so lang wie nötig ().map().spli
- Verständliche Variablennamen
  - → let var1
- unnötige Kommentare vermeiden

CLEAN CODE

- Verständlichkeit / Lesbarkeit
  - Variablen / Funktionsnamen sprechend
  - Code Formatierung
  - Redundante Klammern
  - Ternary Operator bewusst verwenden
- Single Responsibility
- LINTERS / Tool Unterstützung
  - Pre-Commit Hooks
  - CI Checks

Flips 2


1. KISS
2. Sinnvolle Kommentare
3. Styleguide
4. Use your Brain (do not copy from others)

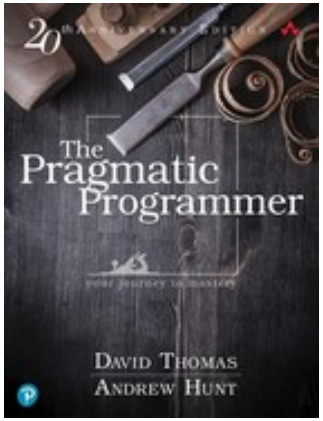

* sprechende Namen
* Redundanzen vermeiden (DRY)
* const before let
don't use var
* keine langen Methoden
* keine Magic Numbers
* Übersichtlichkeit
* Built-in Functions
* Single Responsibility

BEYOND CLEAN CODE

Clean Coder

The Clean Coder: A Code of Conduct for Professional

- **1. Professionalism**
- **2. Saying No**
- **3. Saying Yes**
- **4. Coding**
- **5. Test Driven Development**
- **6. Practicing**
- **7. Acceptance Testing**
- **8. Testing Strategies**
- **9. Time Management**
- **10. Estimation**
- **11. Pressure**
- **12. Collaboration**
- **13. Teams and Projects**
- **14. Mentoring, Apprenticeship, and Craftsmanship**

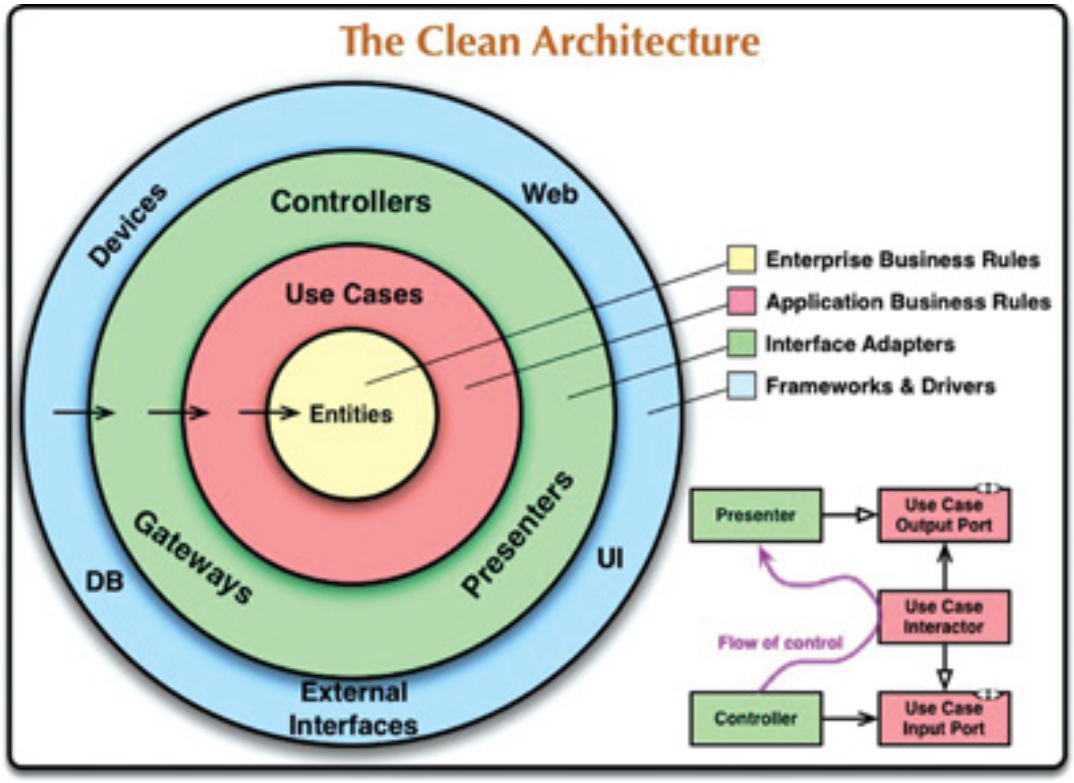


The Pragmatic Programmer:
your journey to mastery,
20th Anniversary Edition,
2nd Edition
(David Thomas, Andrew Hunt;
Addison-Wesley 2019)

Clean Architecture

Clean Architecture:
A Craftsman's Guide
to Software

- **Independent of frameworks**
- **Testable. The business rules can be tested without the UI**
- **Independent of the UI. The UI can change easily, without changing the rest of the system. (The Web is a detail)**
- **Independent of the database (DB is a detail)**



More Links (Generell)

- **ryanmcdermott/clean-code-javascript: Clean Code concepts adapted for JavaScript**
  https://github.com/ryanmcdermott/clean-code-javascript
- https://levelup.gitconnected.com/javascript-refactoring-tips-making-functions-clearer-and-cleaner-c568c299cbb2
- https://codeburst.io/top-10-javascript-errors-from-1000-projects-and-how-to-avoid-them-2956ce008437
- https://www.bbv.ch/images/bbv/pdf/downloads/V2_Clean_Code_V3.pdf
- http://www.brookdalecomp166.com/books/javascript/javascript-best-practice.pdf
- https://blog.logrocket.com/12-tips-for-writing-clean-and-scalable-javascript-3ffe30abfe20/
- https://medium.com/@severinperez/maintainable-code-and-the-open-closed-principle-b088c737262
- https://github.com/abiodunjames/Awesome-Clean-Code-Resources/blob/master/readme.md#javascript
- https://gist.github.com/nicholasren/93e6d49f62ad8ed0b596

More Links (Framework-spezifisch)

- https://itnext.io/clean-code-checklist-in-angular-%EF%B8%8F-10d4db877f74
- https://www.opencodez.com/reactjs/reactjs-best-practices.htm
- https://developers.caffeina.com/clean-code-saves-devs-the-caffeina-approach-to-reactjs-1b56ad15aa64

CODEWARS BEISPIEL STOLZE

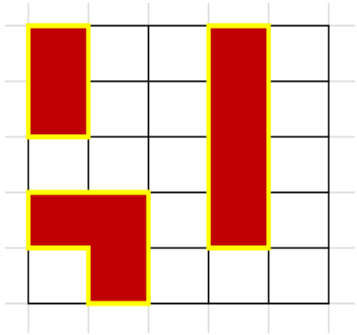
Example Code: Codewars Kata “Land Perimeter”

Task:

Given an array `arr` of strings complete the function `landPerimeter` by calculating the total perimeter of all the islands. Each piece of land will be marked with `'X'` while the water fields are represented as `'O'`. Consider each tile being a perfect `1 x 1`piece of land. Some examples for better visualization:

```
['XOOXO',
 'XOOXO',
 'OOOXO',
 'XXOXO',
 'OXOOO']
```

or

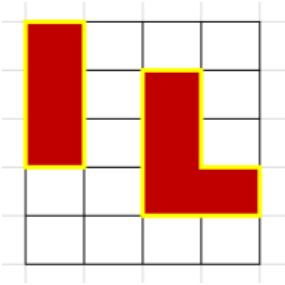


should return: "Total land perimeter: 24",

while

```
['XOOO',
 'XOXO',
 'XOXO',
 'OOXX',
 'OOOO']
```

Or



should return:
"Total land perimeter: 18"

Example Code: Codewars Kata “Land Perimeter”: “Smart” Solution 1

```
function landPerimeter(arr) {
  var n = arr.length - 1
      ,   m = arr[0].length - 1
      ,   p = 0;

  for (var i = 0; i <= n; i++)
    for (var j = 0; j <= m; j++)
      if (arr[i][j] === 'X') {
        p += i === 0 || arr[i - 1][j] === 'O';
        p += j === 0 || arr[i][j - 1] === 'O';
        p += i === n || arr[i + 1][j] === 'O';
        p += j === m || arr[i][j + 1] === 'O';
      }

  return 'Total land perimeter: ' + p;
}
```

Example Code: Codewars Kata “Land Perimeter”: “Smart” Solution 2

```
function landPerimeter(m,C=0) {
  m.forEach((n,i)=>
      [...n].forEach((_,j)=> {
        if (m[i][j] === 'X')
          for (let [r,c] of [[i+1,j],[i,j+1],[i-1,j],[i,j-1]])
            if (!m[r] || m[r][c] !== 'X') C++
      })
  )
  return 'Total land perimeter: '+C
}
```

Example Code: Codewars Kata “Land Perimeter”: “Cleaner” (?) Solution Stolze

```
function landPerimeter(arr) {
  const maxX = arr[0].length - 1;
  const maxY = arr.length - 1;

  const isBetween = (testNr, lowerbound, upperBound) =>
      testNr >= lowerbound && testNr <= upperBound;

  const hasLandAtPosition = (x, y) =>
      isBetween(x, 0, maxX) && isBetween(y, 0, maxY) && arr[y][x] === "X";

  const countNeighbor = (posX, posY, directionX, directionY) =>
      hasLandAtPosition(posX + directionX, posY + directionY) ? 1 : 0;

  const neighborCount = (posX, posY) =>
      countNeighbor(posX, posY, -1, 0) +
      countNeighbor(posX, posY, 1, 0) +
      countNeighbor(posX, posY, 0, -1) +
      countNeighbor(posX, posY, 0, 1);

  const borderCount = (posX, posY) => 4 - neighborCount(posX, posY);

  let perimeter = 0;
  for (let x = 0; x <= maxX; x++) {
    for (let y = 0; y <= maxY; y++) {
      if (hasLandAtPosition(x, y)) {
        perimeter += borderCount(x, y);
      }
    }
  }
  return `Total land perimeter: ${perimeter}`;
}
```

Clean Coding JS

- **Ablauf**
  - Gruppendiskussion (2er Team)
    - Ca. 2-3 Kriterien definieren (Smells, oder Best Practices) mit Beispielen
  - Gruppendiskussion (2x2er Team)
    - Ca. 4 wichtigste Kriterien definieren (Smells, oder Best Practices) mit Beispielen
  - Präsentationen
  - Diskussion (Plenum)